I, Tadahiko Itoh, a Patent Attorney of Tokyo, Japan having my office at 32nd Floor, Yebisu Garden Place Tower, 20-3 Ebisu 4-Chome, Shibuya-Ku, Tokyo 150-6032, Japan do solemnly and sincerely declare that I am the translator of the attached English language translation and certify that the attached English language translation is a correct, true and faithful translation of Japanese Patent Application No. 2004-286042 to the best of my knowledge and belief.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Tadahiko ITOH
Patent Attorney
ITOH International Patent Office
32nd Floor,
Yebisu Garden Place Tower,
20-3 Ebisu 4-Chome, Shibuya-Ku,
Tokyo 150-6032, Japan

(Document Name)	Application for Patent
(Reference Number)	200402839
(Date of Submission)	September 30, 2004
(Destination)	Commissioner of Patent Office
(IPC)	G 06F 17/30
(Inventor)	
(Residence or Address)	c/o RICOH COMPANY, LTD. 3-6, Nakamagome 1-chome, Ohta-ku, Tokyo, Japan
(Name)	Takamitsu Yamada
(Applicant for Patent)	
(Identification Number)	000006747
(Name)	RICOH COMPANY, LTD.
(Attorney)	
(Identification Number)	100102587
(Patent Attorney)	
(Name)	Masayuki Watanabe
(Appointed Attorney)	
(Identification Number)	100077274
(Patent Attorney)	
(Name)	Masatoshi Isomura
(Indication of Official Fees)	
(Prepayment Ledger Number)	068262
(Amount Paid)	¥ 16,000
(List of Submitted Documents)	
(Document Name)	Claims 1
(Document Name)	Specification 1
(Document Name)	Drawing 1
(Document Name)	Abstract 1
(Number of General Power of Attorney)	9808799

CLAIMS

1. An assertion generating system that generates an assertion description which is used for assertion verification of a semiconductor integrated circuit, comprising:

a specification inputting unit that generates design data or specifications and a document for confirming a specification of the semiconductor integrated circuit by graphically editing the specification of the semiconductor integrated circuit based on user operations;

a first storing unit that stores the design data generated by the specification inputting unit;

a property generating unit that generates a property to be verified with respect to the specification of the semiconductor integrated circuit by reading the design data generated by the specification inputting unit from the first storing unit and using the read design data;

a second storing unit that stores the property generated by the property generating unit; and

an assertion generating unit that converts the property into an assertion description by reading the property generated by the property generating unit from

the second storing unit.

2. The assertion generating system as claimed in claim 1, wherein

the specification inputting unit includes a design data generating unit that generates the design data of the semiconductor integrated circuit by editing the specification of the semiconductor integrated circuit with the use of a state transition table or a state transition figure based on user operations.

3. The assertion generating system as claimed in claim 1, wherein

the specification inputting unit includes a design data generating unit that generates the design data of the semiconductor integrated circuit by editing a process sequence of the semiconductor integrated circuit into a timing chart or a time series figure based on user operations.

4. The assertion generating system as claimed in any one of claims 1 through 3, wherein

the specification inputting unit includes a design data generating unit that generates the design data of the semiconductor integrated circuit by editing

the specification of the semiconductor integrated circuit with the use of a logic table or a state table based on user operations.

5. The assertion generating system as claimed in claim 4, wherein

the assertion generating unit converts the property into an assertion description in which an assertion name composed of a table name or a table row number of the logic table or the state table, or a signal name or a state name in the logic table or the state table edited by the specification inputting unit is added.

6. The assertion generating system as claimed in any one of claims 1 through 5, wherein

the specification inputting unit loads the design data into a graphic structure and inputs the graphic structure in the first storing unit.

7. A program that makes a computer function as the units of the assertion generating system as claimed in any one of claims 1 through 6.

8. A circuit verifying system, comprising:

the assertion generating system as claimed in

any one of claims 1 through 6, wherein

the circuit verifying system executes assertion verification of a semiconductor integrated circuit by using the assertion description generated by the assertion generating system.

9. An assertion generating method that generates an assertion description which is used for assertion verification of a semiconductor integrated circuit by a computer which has a program, comprising:

a specification inputting step that generates design data of the semiconductor integrated circuit by graphically editing a specification of the semiconductor integrated circuit based on user operations and inputs the design data in storage;

a property generating step that reads the design data generated at the specification inputting step from the storage and generates a property to be verified with respect to the specification of the semiconductor integrated circuit using the read design data and inputs the property in the storage; and

an assertion generating step that reads the property generated at the property generating step from the storage and converts the property into an assertion description.

10. The assertion generating method as claimed in claim 9, wherein

the property generating step generates at least one or more properties, and

the specification inputting step generates the design data of the semiconductor integrated circuit by editing the specification of the semiconductor integrated circuit with the use of a state transition table or a state transition figure based on user operations.

11. The assertion generating method as claimed in claim 9, wherein

the specification inputting step generates the design data of the semiconductor integrated circuit by editing a process sequence of the semiconductor integrated circuit into a timing chart or a time series figure based on user operations.

12. The assertion generating method as claimed in any one of claims 9 through 11, wherein

the specification inputting step generates the design data of the semiconductor integrated circuit by editing the specification of the semiconductor integrated circuit with the use of a logic table or a state table

based on user operations.

13. The assertion generating method as claimed in claim 12, wherein

the assertion generating step converts the property into an assertion description in which an assertion name composed of a table name or a table row number of the logic table or the state table, or a signal name or a state name in the logic table or the state table edited at the specification inputting step is added.

14. The assertion generating method as claimed in any one of claims 9 through 13, wherein

the specification inputting step loads the design data into a graphic structure and inputs the graphic structure in the storage.

TITLE OF THE INVENTION

ASSERTION GENERATING SYSTEM, CIRCUIT VERIFYING SYSTEM, PROGRAM THEREOF, AND ASSERTION GENERATING METHOD

TECHNICAL FIELD

The present invention relates to a technology that executes circuit verification of an LSI (large scale integration) and so on by using a process based on a

computer program; and in particular, a verifying technology in which design violation and missing verification are confirmed in design data by converting a property of a circuit to be verified into an assertion description and inputting the assertion description to a simulator when the design data of the circuit in a RTL (register transfer level) are dynamically verified by the simulator.

The property to be discussed below is a plain text that defines operations intended and expected in a design circuit to be verified. For example, the property defines a relation between reception and response of a request signal and an approval signal in an arbiter circuit on time series constraint, and defines a regular state of the time series constraint and circuit specifications to be forbidden. In addition, the property is an event to be monitored as a function coverage point which confirms whether a specific circuit sequence has been tested by a simulation. Property verification verifies whether a design description satisfies a predetermined property.

BACKGROUND ART

Recently, a system has been revised or a revised edition of a system has been published frequently,

caused by missing verification of a corner case due to the complexity of a circuit and an increase of circuit size of a semiconductor integrated circuit, by missing confirmation of interface specifications among blocks allocated to plural designers due to group work, or by specification errors of a reusable core purchased from a third party.

These problems come from, for example, insufficient circuit verification or a test scenario that can extract bugs, which were not able to be formed because a corner case and core specifications of the third party were not understood sufficiently.

In order to solve these problems, an assertion verification technology, in which a property with respect to circuit specifications is converted into an assertion description and the assertion description is input to a simulator (computer) for verification, and a function coverage report for showing a warning for assertion violation during the simulation and showing whether a specific circuit sequence has been tested by a simulator is provided, has been proposed and has been recently put into use in actual designing.

An assertion is a note in which intention (in some cases, this shows property) of designing is written and is generally written by a comment sentence in a RTL.

The intention is interpreted by a simulator during the verification, for example, an error log is generated in a case where a circuit to be verified operates differently from the intention. The intention of designing is, for example, an assumption and a precondition of circuit input, and in addition to these, expected operations when the conditions are satisfied.

The simulator checks the following two matters.

(1) Specific events such as an assumption and a precondition have occurred.

(2) Expected operations at that time complete normally.

From the above check (1), the assertion gives feedback of determining information as to whether a specific circuit function is confirmed by verification. Therefore, when the assertion is comprehensively installed in a circuit to be verified, function coverage of the assertion can be obtained; consequently, its verification accuracy can be obtained quantitatively.

From the above check (2), the feedback to a verification debugging can be obtained by the assertion. For example, an assertion installed in a position, from which an influence caused by a circuit malfunction cannot be observed by external terminals, complements a malfunction state. In addition, when conditions such as

an assumption and a precondition are satisfied in any stimulus, this expected operation is checked at any time. Therefore, for example, even when the stimulus is structured by a random system, self-collation with the expected operation, that is, comparison of expected values, can be executed. In addition, an assertion installed in a lower block is also effective in chip level verification.

The intention of designing may be described by a Verilog-HDL for verification being different from a RTL for logic synthesis; however, here, another assertion language is described.

At present, the most popular assertion language is a PSL (property specification language) described in Non-Patent Document 1. The PSL is an assertion language which will be given to IEEE from a standardization organization named Accellera, and is actually a standard language. In the following, an actual example of an assertion description using the PSL is shown.

For example, it is assumed that the following design intention (property) for memory control is desired to be monitored by verification.

1) read_n and write_n do not become Low at the same time

2) at the fall of write_n, enable_n is High

3) at the fall of read_n, enable_n is Low

The PSL description (assertion description) to be given to a simulator as an assertion for the intention is as follows.

```
// psl property memcont1 = never (!write_n && !read_n)
@(posedge clk) ;
// psl assert memcont1 ;
// psl property memcont2 = always (enable_n) @(negedge
write_n) ;
// psl assert memcont2 ;
// psl property memcont3 = always (!enable_n) @(negedge
read_n) ;
// psl assert memcont3 ;
```

All of "psl, property, never, assert, always" in the above PSL description (assertion description) are reserved words in the PSL. All rows start with "//" which shows a comment of the Verilog-HDL. In this, instead of using "//", all of the PSL descriptions can be closed with "/**/".

The words "memcont1,2,3, !write_n && !read_n, posedge clk, enable_n, negedge write_n, !enable_n, negedge read_n" are descriptions by a user definition, and actual installed data names, that is, memory control signal names which appear in the RTL for defining the intention of 1) through 3) except for "memcont1,2,3".

In addition, "memcont1 through 3" are assertion

names and are used for knowing an error log and a function coverage being given from the simulator as feedback.

The assertion description by the PSL has the following structural elements.

// psl property <an assertion name> =

<an event to be monitored> -> <expected operation when a condition is satisfied> @<a strobe condition> ;

// psl assert <an assertion name> ;

For example, in the assertion of 1), "memcont1" is an assertion name, "!write_n && !read_n" is an event to be monitored, in this case, "never" is added in front of the event; therefore, it is monitored that the condition in ( ) never becomes true (if "always" is added, it is monitored that the condition in ( ) is true). By "@(posedge clk)" which follows the above statement, it is defined that monitoring the condition is executed at every time of rising edges. "->" changes to "|=>" or "|->" depending on the time when the expected operation is started. In this, the above example does not accompany expected operations.

The inside of ( ) of each part except the strobe condition is described by the Verilog-HDL which returns Boolean values. The inside of ( ) of the strobe

condition has a description form of a sensitivity list described by an "always sentence" of the Verilog-HDL.

In the PSL, an event to be monitored can be defined, and each part of expected operations at the time when the condition is satisfied can define a sequence, that is, the changes of events and expected operations in plural cycles can be defined. An example is explained.

The intention of designing: When a clear start condition (m_task == 2'b00) is satisfied in a clearing process of a memory, in order to initialize all words (256 words), a writing operation is repeated 256 times. Here, one writing operation is completed by the operation of Low→High→High of write_n synchronized with a rise cycle (positive cycle) of clk.

The PSL description:

```
// psl sequence WRITE_PULSE = { !write_n; write_n;
write_n };
// psl property CLEAR_MEM_WRITE_N =
// always { m_task == 2'b00 } |=> { WRITE_PULSE [*256] }
// @(posedge clk) ;
// psl assert CLEAR_MEM_WRITE_N;
```

"sequence" defines a Low→High→High sequence of "write_n" by a macro-definition.

Here, it is the part of expected operations which follows "|=>" that is to be focused on. When

"m_task == 2'boo", the expected operation is WRITE_PULSE, that is, it is expected that the sequence of Low→High→High being three cycles of write_n be repeated by 256 times.

However, in such an assertion verifying technology, for example, in a case where the assertion description language is written by a manual input when the specifications are defined, there is a possibility that a mistake may be made in the assertion itself. As a result, a report of the warning of the assertion violation and the function coverage obtained by the assertion verification cannot be fed back to prevent the system revision and the revised edition of the system. Further, this requires a verification period for debugging the assertion description language. Consequently, the verification efficiency becomes worse.

In other words, the major premise of the assertion verifying technology is that the circuit specifications and the assertion description match each other; however, there is no method to confirm the matching before executing the simulation. Further, with respect to a complex finite state machine, it is difficult to describe a sequence of a comprehensive state transition by manual operation.

As conventional technologies with respect to

the assertion verification, for example, there are technologies disclosed in Patent Document 1 and Patent Document 2.

In Patent Document 1, a technology, in which a state transition machine is modeled by a graph and a state transition path that satisfies the assertion is searched, is disclosed. Further, a technology to confirm the rightness of the assertion by only an analysis of this graph is disclosed. By the technologies, matching the circuit specifications with the assertion description can be confirmed.

However, in these technologies, the state transition machine which is made to be a graph manually forms a model of the state transition machine which becomes a basic model, or the state transition machine is extracted from design data formed by a description language of a RTL. Consequently, mixing a mistake in an object to be matched, that is, in the model of the state transition machine which should be the circuit specifications, cannot be avoided.

In addition, in Patent Document 2, a technology, which generates an assertion for verifying a circuit by extracting a data transfer structure from design data of a RTL and expanding the data transfer structure into a graphic structure, is disclosed. However, in this

technology, since the assertion which is the circuit specifications is extracted from the design data, matching the circuit specifications with the assertion description cannot be assured.

[Patent Document 1] Japanese Laid-Open Patent Application No. 2000-181933

[Patent Document 2] Japanese Laid-Open Patent Application No. 2000-142918

[Non-Patent Document 1] Property Specification Language Reference Manual Version 1.1 June 9, 2004, online, retrieved in September 14, 2004, from the Internet, URL: http://www.eda.org/vfv/docs/PSL-v1.1.pdf

Consequently, in the conventional technologies, matching the circuit specifications with the assertion description cannot be assured.

DISCLOSURE OF THE INVENTION

The object of the present invention is to solve the problems in the conventional technologies, for example, to enable high reliability and high efficiency in assertion verification for an LSI and so on.

In order to achieve the above object, according to the present invention, a property to be verified is automatically converted into an assertion description language, based on a visual state transition figure, a

timing chart, a process sequence chart, and so on used when a designer forms specifications of a circuit.

According to the present invention, since the assertion description of a property to be verified is automatically generated from electronic data of a state transition figure which is attached to the specifications, matching the circuit specification with the assertion is established. Therefore, work to be used for debugging the assertion can be reduced. In addition, since the generated assertion is a function coverage point generated by comprehensive search of paths of the state transition, after the verification, feedback that a path has not been tested can be obtained by a function coverage report; consequently, missing a test of a corner case can be prevented. In addition, since the assertion description of a property to be verified is automatically generated from a selection condition of timing charts and signals which are attached to the specifications, or electronic data of figures or a table form in which a process sequence is noted or defined, matching the circuit specification with the assertion is established. Therefore, work to be used for debugging the assertion can be reduced. Especially, in many cases, the selection condition of timing charts and signals which are attached to the specifications or the process sequence is an

important assertion to be surely tested at the time of verification; however, since the assertion generated in the embodiments is an important property, a designer who does not know the specifications and the circuit functions sufficiently can execute verification by using the key property; as a result, the number of times of revising the system caused by missing the verification can be reduced.

BEST MODE FOR CARRYING OUT THE INVENTION

Next, referring to the drawings, preferred embodiments of the present invention are explained in detail.

FIG. 1 is a block diagram showing a structural example of hardware of which an assertion generating system and a circuit verifying system according to the present invention consist. FIG. 2 is a block diagram showing a first functional structure example of the assertion generating system and the circuit verifying system using the assertion generating system according to the present invention. FIG. 3 is an explanatory diagram showing a memory content example of a graphic structure which is used in the assertion generating system according to the present invention. FIG. 4 is an explanatory diagram showing a state transition figure and

a graphic structure example which is used in the assertion generating system according to the present invention. FIG. 5 is a block diagram showing a second functional structure example of the assertion generating system according to the present invention. FIG. 6 is a block diagram showing a third functional structure example of the assertion generating system according to the present invention. FIG. 7 is an explanatory diagram showing a timing chart example which is used in the assertion generating system according to the present invention. FIG. 8 is an explanatory diagram showing a structural body example for storing information with respect to signals defined on the timing chart and information with respect to timing constraints for the signals which is used in the assertion generating system according to the present invention. FIG. 9 is an explanatory diagram showing a memory content example of property information with respect to the timing chart shown in FIG. 7. FIG. 10 is a block diagram showing a fourth functional structure example of the assertion generating system according to the present invention. FIG. 11 is an explanatory diagram showing a structural example of a logic table of a full adder which is used in the assertion generating system according to the present invention. FIG. 12 is an explanatory diagram showing a

structural body example for showing one cell when the cell in a table is loaded in a memory. FIG. 13 is an explanatory diagram showing a content example when a sheet is formed by combining structural bodies showing the cells shown in FIG. 12. FIG. 14 is an explanatory diagram showing a content when the logic table of the full adder shown in FIG. 11 is loaded in the memory structure shown in FIG. 13. FIG. 15 is an explanatory diagram showing a structural example of a process sequence in a table form which is used in the assertion generating system according to the present invention.

In FIG. 1, the reference number 101 is storage composed of a memory, a hard disk, and so on; the reference number 102 is a CPU (central processing unit); the reference number 103 is an input unit composed of a keyboard, a mouse, and so on; the reference number 104 is a display unit composed of a CRT (cathode ray tube), a LCD (liquid crystal display) and so on; the reference number 105 is a system bus, and a computer system is composed of the units 101 through 105 in which the units 101 through 104 are connected by the system bus 105.

The storage 101 stores functional descriptions of hardware not depending on architecture, and functions and means according to the present invention in feasible program formats which are read from recording media, from

which a computer can read the programs, such as a CR-ROM (compact disc-read only memory), a DVD (digital video disc/digital versatile disc), and a FD (flexible disk). The CPU 102 executes the programs being stored in the storage 101, and executes processes of functions of the assertion generating system having the structure shown in FIG. 2.

A user interactively inputs commands to execute the programs that realize inputting and editing of the functional descriptions of hardware not depending on architecture and means of the present invention by using the input unit 103. The display unit 104 displays the functional descriptions of hardware not depending on architecture, and the progress and the results of the programs, which realize the means of the present invention, executed by the CPU 102.

FIG. 2 shows a basic structure of an assertion generating system 207 and a circuit verifying system using the assertion generating system 207 according to the present invention. First, a circuit specification to be specified in specifications is input from a graphical editor 201 by user operations, and the circuit specification formed by the graphical editor 201 is stored in the storage 101 as specification electronic data 202. In this, as the graphical editor 201, a

graphical editor such as a timing chart editor disclosed in, for example, Japanese Laid-Open Patent Application No. 1-309185 can be used.

Next, a syntax analyzer 203 reads the specification electronic data 202 being stored in the storage 101 and analyses the syntax, and a property extractor 204 extracts a property to be verified from the results of the syntax analysis.

An assertion generator 205 generates an assertion description language 206, which can be input to a simulator or a static property checker, from the property extracted from the property extractor 204. The assertion description language 206 generated in such a manner is stored in the storage 101 and is input to a simulator, and verification is executed.

As mentioned above, according to the assertion generating system of this example, when assertion verification, which verifies design data in which a property with respect to a specification of a semiconductor integrated circuit or a part of the specification of the semiconductor integrated circuit is described in an assertion description with design data of a circuit to be verified of the property, is executed by a simulator or a static verification tool, the graphical editor 201 generates the specification electronic data

202 (design data) by editing a specification of a finite state machine by a visual expression on a state transition table or a state transition figure. With respect to the circuit specification defined at the graphical editor 201, the syntax analyzer 203 and the property extractor 204 generates a property to be verified based on the design data, and the assertion generator 205 outputs this property in an assertion description.

Here, the design data (specification electronic data) formed at the graphical editor 201 are expanded into a graphic structure on the memory, and the property extractor 204 can refer to the graphic structure. Further, as the design data (specification electronic data) formed at the graphical editor 201, data, in which a process sequence of a circuit is shown in a timing chart or a time series figure by using a business tool that can edit a table input or a figure, are included, in addition to the state transition table and the state transition figure.

FIG. 5 shows an example in which the state transition figure is used as the design data (specification electronic data) to be formed at the graphical editor 201. In FIG. 5, graphical information 501 is, for example, the state transition figure

exemplified in FIG. 4(a), and is edited and formed by the graphical editor 201 and is stored in the storage being a hard disk, a system memory, and so on.

The graphical information 501 includes visual information, that is, coordinate information and so on when structural elements of the state transition figure are disposed, and the state transition figure expressed visually can be attached to specifications by printing out.

A graph search engine 502 has functions of the syntax analyzer 203 and the property extractor 204 shown in FIG. 2, and extracts a graphic structure exemplified in FIG. 4(b) from the state transition figure exemplified in FIG. 4(a), and temporarily stores the graphic structure in a system memory. In the extracted graphic structure, the structural elements of the state transition figure are formed as nodes, and the graph search engine 502 searches state transition paths. For example, the graph search engine 502 searches all state transition paths from an initial state of the state transition machine, extracts the paths, and loads them in the system memory or stores them in the hard disk as a path database 503.

A property converter 504 converts the state transition paths searched by the graph search engine 502

into an assertion description language 505 by referring to the path database 503. The assertion description language 505 converted at the property converter 504 is described, for example, as a function coverage point for confirming whether the assertion description language 505 has been tested at a simulator.

A memory content of the graphic structure to be used at the graph search engine 502 is shown in FIG. 3. In FIG. 3, the reference number 301 is a structural body that stores information of each state in the state transition figure, and the reference number 302 is a structural body that stores information of arcs in the state transition figure. These structural bodies 301 and 302 mutually refer to pointers and form a graph in which the respective pointers are made to be nodes.

In FIG. 4, FIG. 4(a) shows an example of a state transition figure, and FIG. 4(b) shows a graphic structure formed from the state transition figure. The graph search engine 502 shown in FIG. 5 searches the state transition paths from the state transition figure shown in FIG. 4(a) by using a path search in graph logic, for example, an algorithm of a shortest path search. As a result, the graphic structure shown in FIG. 4(b) is formed and the graphic structure is stored in the path database 503.

Next, an example of a state transition path to which can be stored in the path database 503 shown in FIG. 5 is shown by using the example shown in FIG. 4.

```
init ;
(GO) S1 ;
init ;
```

In this example, a path from "init" to "S1" is searched, and when "GO" is true, that is, is "1", the state transition path is formed.

In addition, an assertion description by PSL, which is generated by the property converter 504 in FIG. 5 (the assertion generator 205 in FIG. 2) from this state transition path, is shown as follows.

```
property init_to_S1 = always { state == init & GO} |=>
{ state == S1} ;
```

In case of a state transition machine, it is necessary whether all states are visited in verification, in order to quantify the verification accuracy of important functions being keys of circuit functions. Whether each state of the state transition machine is visited can be determined by confirming the function coverage of state verification by cover syntax of PSL. In case of the example shown in FIG. 4, it is enough that only state names stored in nodes of a graph are made to be a list by a cover description of PSL.

In the following, the result is shown. In this, "state" is a state register name for holding the state at the time when a state transition machine is installed.

```
// psl cover ( state == init) ;
// psl cover ( state == S1) ;
```

Here, in a case where the graphical information 501 shown in FIG. 5 has information with respect to a graphic structure with the coordinate information, as shown in FIG. 3 and FIG. 4(a), the graph search engine 502 directly refers to the graphic structure in the graphical information 501.

Next, referring to FIG. 6, another embodiment of the present invention is explained. In the embodiment shown in FIG. 6, timing chart information to be attached to specifications is extracted from timing chart electronic data 601, and the interrelation with respect to timing of a signal group to be verified is made to be an assertion description.

In FIG. 6, the reference number 602 is a timing chart reader that reads the timing chart electronic data 601, the reference number 603 is a timing property extractor that extracts an interrelation with respect to timing of a signal group to be verified (timing chart information) and stores the interrelation in a timing chart database 604, and the reference number 605 is a

property converter that reads the timing chart information from the timing chart database 604 and generates an assertion description language 606 that can be input to a simulator or a static property checker.

In this, the timing chart electronic data 601 can be data formed and edited by a timing chart editor, in this case, the timing chart reader 602 becomes a syntax analyzer for timing chart information formed and edited by the timing chart editor.

The memory content which is used by the timing property extractor 603 is shown in FIG. 8. In FIG. 8, the reference number 801 is a structural body that stores information with respect to signals to be defined on the timing chart, and the reference number 802 is a structural body that stores information with respect to timing constraint for the signals.

An actual example of the timing chart that is read by the timing chart reader 602 is shown in FIG. 7. In the example shown in FIG. 7, "ack" rises during 2 to 5 cycles from the rise of "req" synchronized with "clk".

In the example shown in FIG. 7, the timing constraint (property), in which the rise of "ack" must be received in the range of 2 to 5 of "clk" after the output signal of "req" becomes 1, is described.

The timing property extractor 603 shown in FIG.

6 extracts the timing information (property) to be verified by using signal attributes such as IN, OUT, and CK, and "pos" that means a rising condition, and interrelation with respect to timing of a signal group such as "<-" and "->" as keywords.

The timing chart information shown in FIG. 7 formed and edited by the timing chart editor is stored in hardware by being edited in, for example, the following list. In the following list, 1 through 3 rows show pins and input/output attributes, for example, a clock input, and 4 through 7 rows show waveform information. In the sixth row, a property, in which "ack" rises during 2 to 5 cycles from the rise of "req" synchronized with "clk", is shown.

```
1 CK      clk;
2 IN      ack;
3 OUT     req;
4 {clk,req,ack} = {1,0,0};
5 {clk,req,ack} = {1,1,0};
6 {clk,req,ack} = {1,0,0} [2:5];
7 {clk,req,ack} = {1,0,1};
```

A memory content of property information with respect to timing loaded by this timing chart information is shown in FIG. 9.

Next, an example, in which a business tool

capable of inputting a table and editing a figure is used for editing a logic table and a time series property of the above circuit, is explained.

FIG. 11 is a structural example in which a logic table of a full adder is input by Excel (a registered trademark) being spread sheet software of Microsoft Corp. NAME, CLOCK, CONDITION, and EXPECT shown in a bold type are reserved words when a table definition is executed, and show a property name, a clock name, a start condition of an assertion check, and expected operations when the assertion check is started, respectively.

In the right under row of CONDITION and EXPECT, signal names are described, and under the names, logic values are entered. According to Excel, a sheet is composed of cells, a cell is a basic unit of a table, and the sheet can be output as an ASCII file by separating cells with ",".

In the case of FIG. 7, the signal name, the 1/2 pulse of clock, and so on are managed by cell, and after loading in a memory, the cell structure is scanned and the memory content shown in FIG. 9 is obtained.

FIG. 12 is a structural body for expressing one cell when a cell of Excel is loaded in a memory. Members of top, bottom, left, and right in this structural body

respectively execute a pointer reference to up and down and left and right.

FIG. 13 is a diagram showing a structural example of a sheet in which structural bodies of the cells shown in FIG. 12 are combined. The reference number 1301 is a pointer showing a sheet, and is composed of "next" being a pointer reference of the next sheet and a member that executes a pointer reference for a header of a cell. The reference number 1302 is a pointer of a cell, and executes a pointer reference for up and down and left and right by using the members of top, bottom, left, and right. By the above method, scanning cells on the sheet can be easily executed in a memory, and the search by the keywords such as NAME, CLOCK, CONDITION, and EXPECT and the interpretation of the logic table can be executed.

FIG. 14 shows a content when the logic table of the full adder shown in FIG. 11 is loaded in the memory structure shown in FIG. 13.

In the following, a PSL description, which generates an assertion description from the logic table of the full adder shown in FIG. 11, is shown. In this, the table and the verification data (that is, assertion) can be easily linked by adding the row number of the table to the structure of the assertion name. For

example, on the first row, FADD_S_0__FADD_csv_line_5 is the assertion name, and line_5 corresponds to the row number of the table.

```
psl property FADD_S_0__FADD_csv_line_5 = always {!A & !B
& !CI} |-> {S === 1'b0};
psl property FADD_CO_0__FADD_csv_line_5 = always {!A & !B
& !CI} |-> {CO === 1'b0};
psl property FADD_S_1__FADD_csv_line_6 = always {!A & !B
& !CI} |-> {S === 1'b1};
psl property FADD_CO_1__FADD_csv_line_6 = always {!A & !B
& !CI} |-> {CO === 1'b0};
psl property FADD_S_2__FADD_csv_line_7 = always {!A & !B
& !CI} |-> {S === 1'b1};
psl property FADD_CO_2__FADD_csv_line_7 = always {!A & !B
& !CI} |-> {CO === 1'b0};
psl property FADD_S_3__FADD_csv_line_8 = always {!A & !B
& !CI} |-> {S === 1'b0};
psl property FADD_CO_3__FADD_csv_line_8 = always {!A & !B
& !CI} |-> {CO === 1'b1};
psl property FADD_S_4__FADD_csv_line_9 = always {!A & !B
& !CI} |-> {S === 1'b1};
psl property FADD_CO_4__FADD_csv_line_9 = always {!A & !B
& !CI} |-> {CO === 1'b0};
psl property FADD_S_5__FADD_csv_line_10 = always {!A & !B
& !CI} |-> {S === 1'b0};
```

```
psl property FADD_CO_5__FADD_csv_line_10 = always {!A
& !B & !CI} |-> {CO === 1'b1};
psl property FADD_S_6__FADD_csv_line_11 = always {!A & !B
& !CI} |-> {S === 1'b0};
psl property FADD_CO_6__FADD_csv_line_11 = always {!A
& !B & !CI} |-> {CO === 1'b1};
psl property FADD_S_7__FADD_csv_line_12 = always {!A & !B
& !CI} |-> {S === 1'b1};
psl property FADD_CO_7__FADD_csv_line_12 = always {!A
& !B & !CI} |-> {CO === 1'b1};
```

In FIG. 10, another embodiment of the present invention is explained. In this embodiment, a sequence to be attached to specifications is stored as table data 1001, a syntax analyzer 1002 loads a sequence which is defined in a table form in a system memory, a sequence extractor 1003 extracts a sequence of a process to be verified, and a property converter 1004 converts the sequence into an assertion description 1005.

An example of a sequence of a process is shown in FIG. 15. This example shows a property equal to that shown in FIG. 7. This example is also formed by Excel. The syntax analyzer 1002 and the sequence extractor 1003 resolve a sequence described in cells into time series, and the property converter 1004 generates an assertion description by using a SERE expanded regular expression

of PSL.

In the resolution of the sequence into time series, ";" in a character string is interpreted as behavior during one clock cycle. In FIG. 15, "0[*2:5]" uses a repetition expression of PSL, and means that a LOW period continues during 2 to 5 cycles. Further, since "ack" is described as "0[*2:5];1;0", "ack" shows behavior that a Low period continues during 2 to 5 cycles and after this High and Low follow.

In the following, an assertion description of PSL generated based on FIG. 15 is shown.

```
psl property reqack_ack_0__REQ_csv_line_5 = always
{(rose)} |-> {{ack === 1'b0}[*2:5];{ack === 1'b1};{ack
=== 1'b0};
```

Programs which the CPU 102 shown in FIG. 1 executes process flows shown in FIGS. 5, 6, and 10 are recorded in a magnetic medium and so on in a feasible form and stored in the storage 101.

As explained by using FIGs. 1 through 15, the assertion generating system 207 according to the embodiment generates an assertion description that is used for assertion verification of a semiconductor integrated circuit. The graphical editor 201 (specification inputting unit) generates design data of the semiconductor integrated circuit by graphically

editing a specification (finite state machine, process sequence) of the semiconductor integrated circuit into a state transition table and a state transition figure or a timing chart and a time series figure based on user operations, forms the specification electronic data 202 by expanding the design data to a graphic structure, and causes the storage to store the data. In addition, the syntax analyzer 203 and the property extractor 204 (property generating unit) read the specification electronic data (design data) from the storage and generate a property to be verified with respect to the specification of the semiconductor integrated circuit based on the design data. Further, the assertion generator 205 (assertion generating unit) converts the property into the assertion description language 206.

In this, in the graphical editor 201 (specification inputting unit), the design data of the semiconductor integrated circuit can be generated by editing the specification of the semiconductor integrated circuit with the use of a logic table or a state table based on user operations. The assertion generator 205 generates an assertion description; in which an assertion name composed of a table name or a table row number of the logic table or the state table edited at the graphical editor 201 (specification inputting unit), or

an assertion name composed of a signal name or a state name in the logic table or the state table is added.

The circuit verification system according to the embodiment provides the assertion generating system 207 and executes assertion verification of the semiconductor integrated circuit by using the assertion description generated by the assertion generating system 207.

As mentioned above, according to the embodiments, since the assertion description of a property to be verified is automatically generated from electronic data of the state transition figure which is attached to the specifications, matching the circuit specification with the assertion is established. Therefore, work to be used for debugging the assertion can be reduced.

In addition, since the generated assertion is a function coverage point generated by comprehensive search of paths of the state transition, after the verification, feedback that a path has not been tested can be obtained by a function coverage report; consequently, missing a test of a corner case can be prevented.

In addition, since the assertion description of a property to be verified is automatically generated from a selection condition of timing charts and signals which

are attached to the specifications, or electronic data of figures or a table form in which a process sequence is noted or defined, matching the circuit specification with the assertion is established. Therefore, work to be used for debugging the assertion can be reduced.

In many cases, the selection condition of timing charts and signals which are attached to the specifications or the process sequence is an important assertion to be surely tested at the time of verification; however, since the assertion generated in the embodiment is an important property, a designer who does not know the specifications and the circuit functions sufficiently can execute verification by using the key property; as a result, the number of times of revising the system caused by missing the verification can be reduced.

While the present invention is described above with reference to specific embodiments shown in FIGs. 1 through 15, it should be apparent that the invention is not limited to these embodiments, but numerous modifications could be made thereto by those skilled in the art without departing from the basic concept and scope of the invention. For example, in the embodiments, programs which realize the property extractor 204, the assertion generator 205, and so on are stored in the

storage 101 by using recording media such as a CD-ROM, a DVD, and a FD; however, the programs can be stored in the storage 101 by down loading via a network using a communication apparatus.

Further, in the embodiments, the assertion description using PSL is explained; however, the embodiments can be applied to other assertion descriptions.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram showing a structural example of hardware of which an assertion generating system and a circuit verifying system according to the present invention consist;

FIG. 2 is a block diagram showing a first functional structure example of the assertion generating system according to the present invention;

FIG. 3 is an explanatory diagram showing a memory content example of a graphic structure which is used in the assertion generating system according to the present invention;

FIG. 4 is an explanatory diagram showing a state transition figure and a graphic structure example which is used in the assertion generating system according to the present invention;

FIG. 5 is a block diagram showing a second functional structure example of the assertion generating system according to the present invention;

FIG. 6 is a block diagram showing a third functional structure example of the assertion generating system according to the present invention;

FIG. 7 is an explanatory diagram showing a timing chart example which is used in the assertion generating system according to the present invention;

FIG. 8 is an explanatory diagram showing a structural body example for storing information with respect to signals defined on a timing chart and information with respect to timing constraints for the signals which is used in the assertion generating system according to the present invention;

FIG. 9 is an explanatory diagram showing a memory content example of property information with respect to the timing chart shown in FIG. 7;

FIG. 10 is a block diagram showing a fourth functional structure example of the assertion generating system according to the present invention;

FIG. 11 is an explanatory diagram showing a structural example of a logic table of a full adder which is used in the assertion generating system according to the present invention;

FIG. 12 is an explanatory diagram showing a structural body example of one cell when the cell in a table is loaded in a memory;

FIG. 13 is an explanatory diagram showing a content example when a sheet is formed by combining the structural bodies showing a cell shown in FIG. 12;

FIG. 14 is an explanatory diagram showing a content example when the logic table of the full adder shown in FIG. 11 is loaded in the memory structurer shown in FIG. 13; and

FIG. 15 is an explanatory diagram showing a structural example of a process sequence in a table form which is used in the assertion generating system according to the present invention.

DESCRIPTION OF REFERENCE NUMBERS

101: storage, 102: CPU, 103: input unit, 104: display unit, 105: system bus, 201: graphical editor, 202: specification electronic data, 203: syntax analyzer, 204: property extractor, 205: assertion generator, 206: assertion description language, 301: structural body (pointer to arc set), 302: structural body (next ark pointer), 501: graphical information, 502: graph search engine, 503: path database (DB), 504: property converter, 505: assertion description language, 601: timing chart

electronic data, 602: timing chart reader, 603: timing property extractor, 604: timing chart database (DB), 605: property converter, 606: assertion description language, 801: structural body (pointer to relating signal), 802: structural body (next constraint pointer), 1001: table data, 1002: syntax analyzer, 1003: sequence extractor, 1004: property converter, 1005: assertion description language, 1301: pointer (showing a sheet), and 1302: pointer (of a cell)

ABSTRACT

In order to solve a problem that matching a circuit specification with an assertion description is not assured in a conventional technology, the present invention enables to establish high reliability and high efficiency in assertion verification for an LSI and so on.

In an assertion generating system 207, a graphical editor (specification inputting unit) 201 generates design data (specification electronic data 202) of a semiconductor integrated circuit by graphically editing a specification (finite state machine, process sequence) of the semiconductor integrated circuit with the use of a state transition table and a state transition figure or a timing chart and a time series figure based on user operations, and a property generating unit composed of a syntax analyzer 203 and a property extractor 204 generates a property to be verified with respect to the specification of the semiconductor integrated circuit based on the design data. An assertion generator 205 converts the property into an assertion description language 206.

FIG. 2

FIG.1

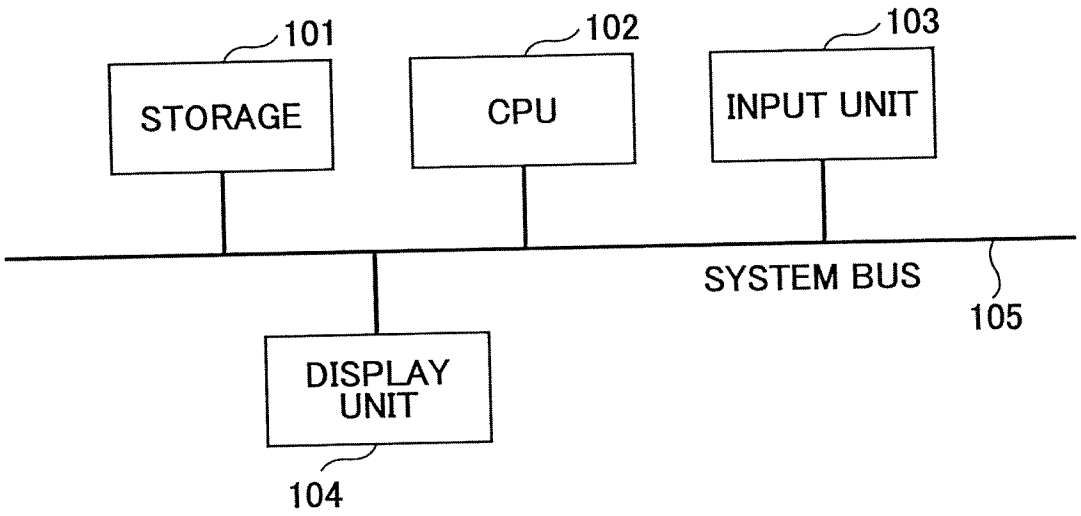
101
STORAGE
102
CPU
103
INPUT UNIT
SYSTEM BUS
105
DISPLAY UNIT
104

FIG.2

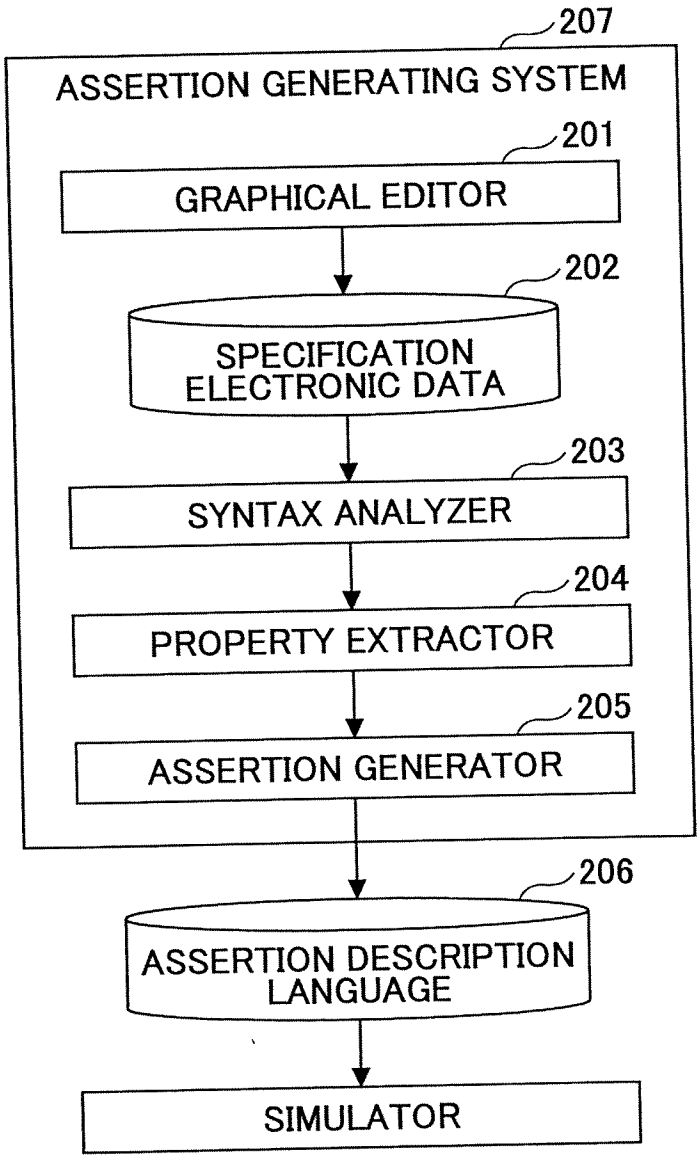
207
ASSERTION GENERATING SYSTEM
201
GRAPHICAL EDITOR
202
SPECIFICATION ELECTRONIC DATA
203
SYNTAX ANALYZER
204
PROPERTY EXTRACTOR
205
ASSERTION GENERATOR
206
ASSERTION DESCRIPTION LANGUAGE
SIMULATOR

FIG.3

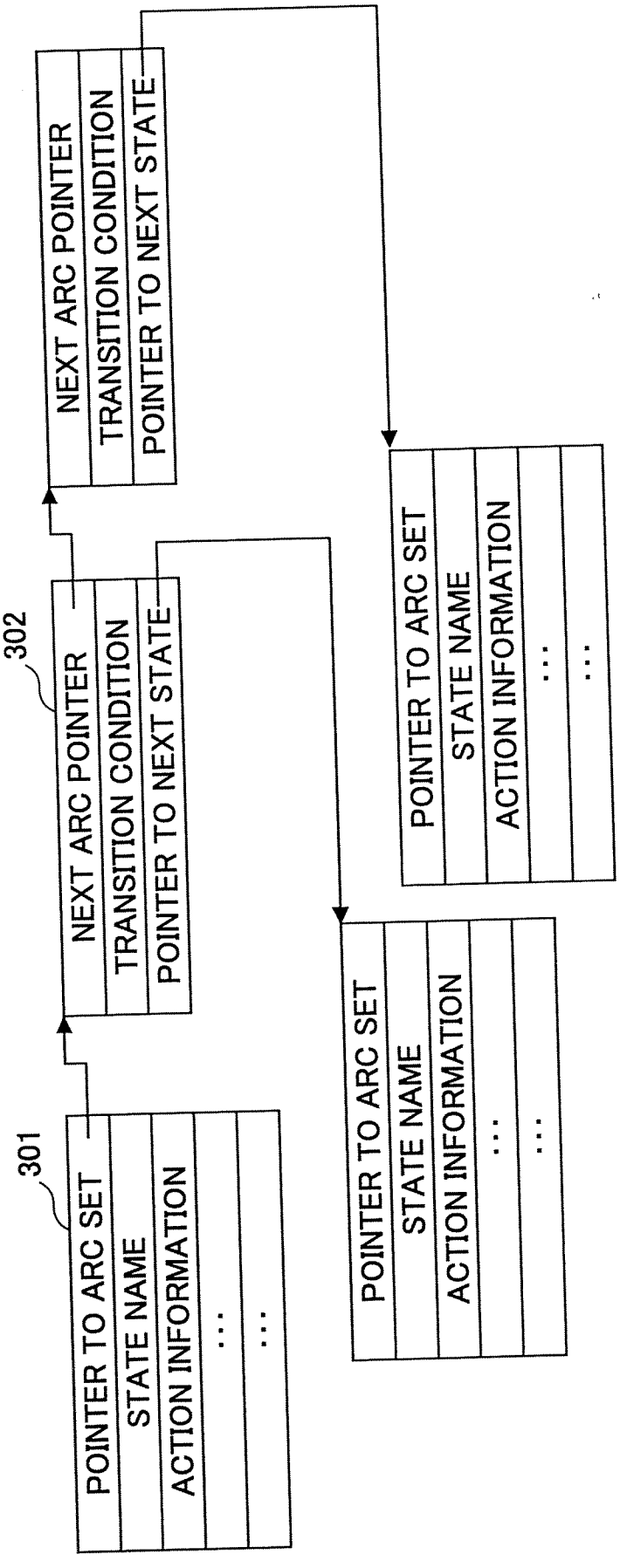
301
POINTER TO ARC SET
STATE NAME
ACTION INFORMATION
...
...
302
NEXT ARC POINTER
TRANSITION CONDITION
POINTER TO NEXT STATE
NEXT ARC POINTER
TRANSITION CONDITION
POINTER TO NEXT STATE
POINTER TO ARC SET
STATE NAME
ACTION INFORMATION
...
...
POINTER TO ARC SET
STATE NAME
ACTION INFORMATION
...
...

FIG.4

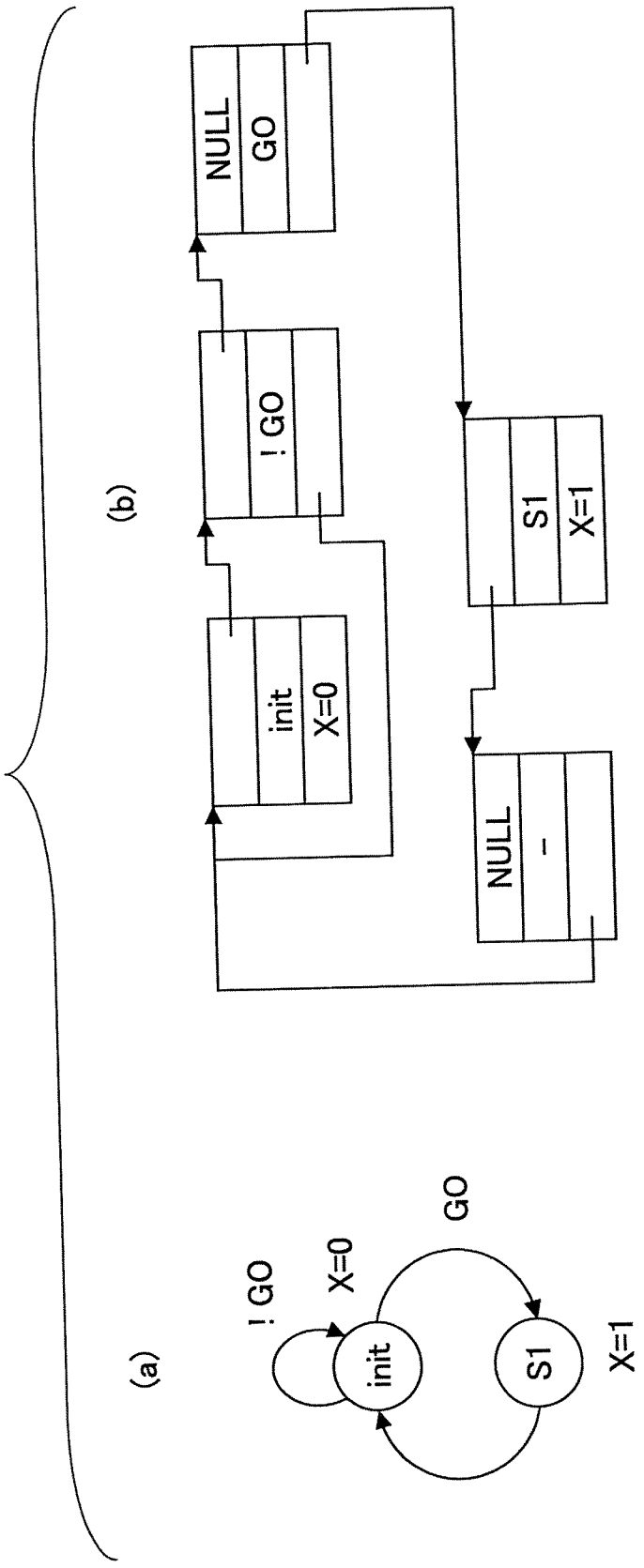

FIG.5

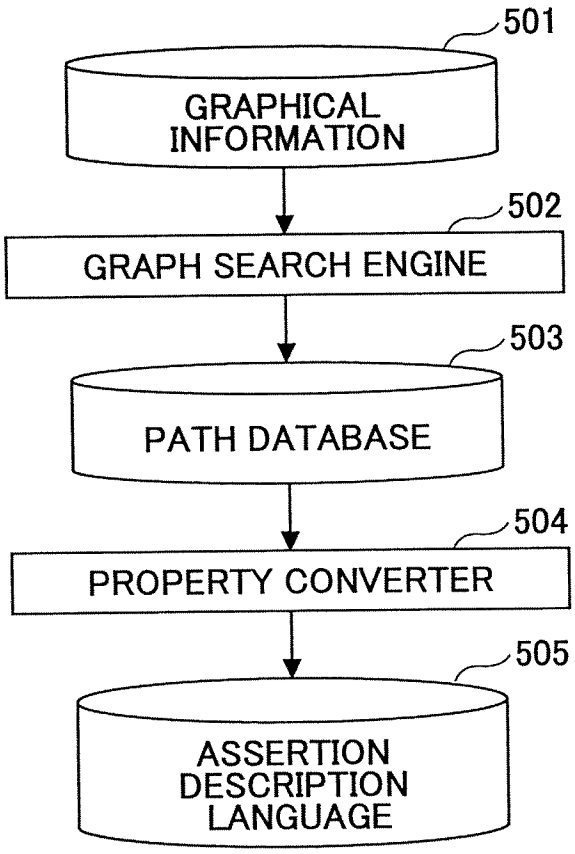
501
GRAPHICAL INFORMATION
502
GRAPH SEARCH ENGINE
503
PATH DATABASE
504
PROPERTY CONVERTER
505
ASSERTION DESCRIPTION LANGUAGE

FIG.6

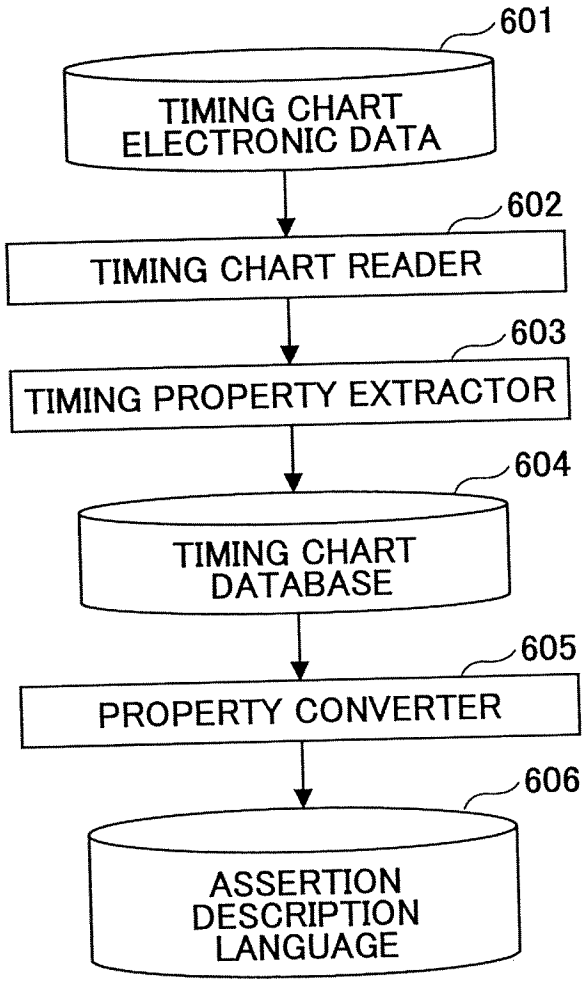
601
TIMING CHART ELECTRONIC DATA
602
TIMING CHART READER
603
TIMING PROPERTY EXTRACTOR
604
TIMING CHART DATABASE
605
PROPERTY CONVERTER
606
ASSERTION DESCRIPTION LANGUAGE

FIG.7

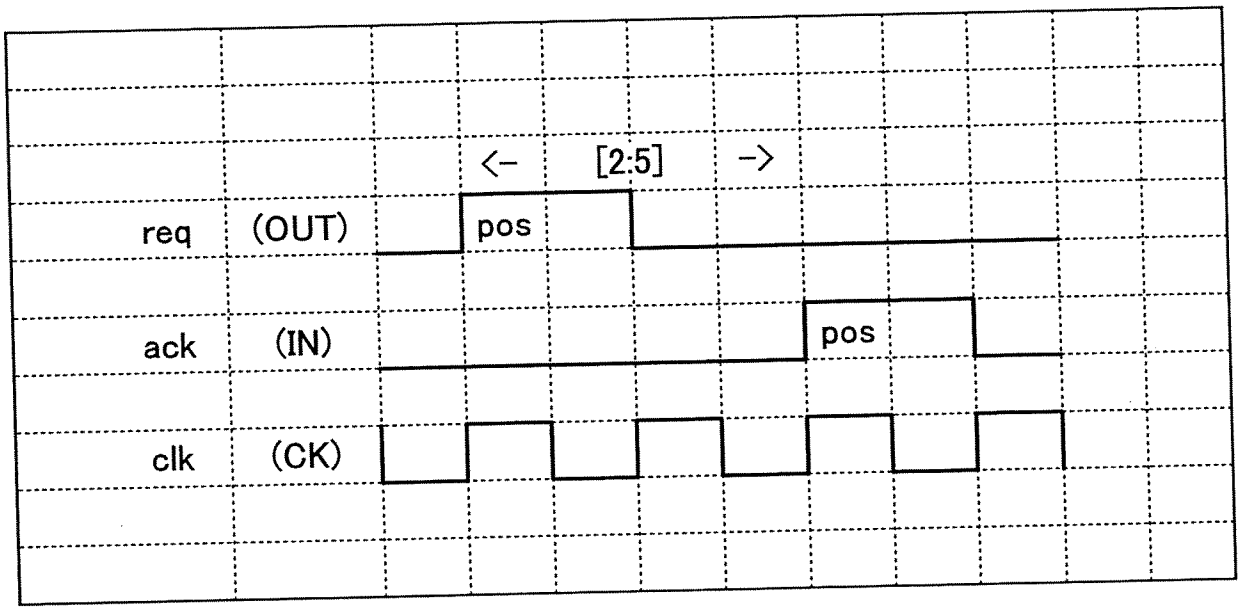
<-
[2:5]
->
req
(OUT)
pos
ack
(IN)
pos
clk
(CK)

FIG.8

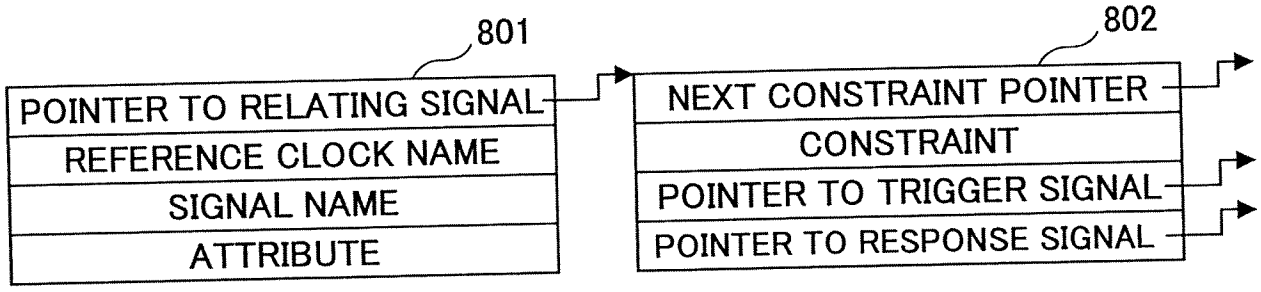
801
802
POINTER TO RELATING SIGNAL
REFERENCE CLOCK NAME
SIGNAL NAME
ATTRIBUTE
NEXT CONSTRAINT POINTER
CONSTRAINT
POINTER TO TRIGGER SIGNAL
POINTER TO RESPONSE SIGNAL

FIG.9

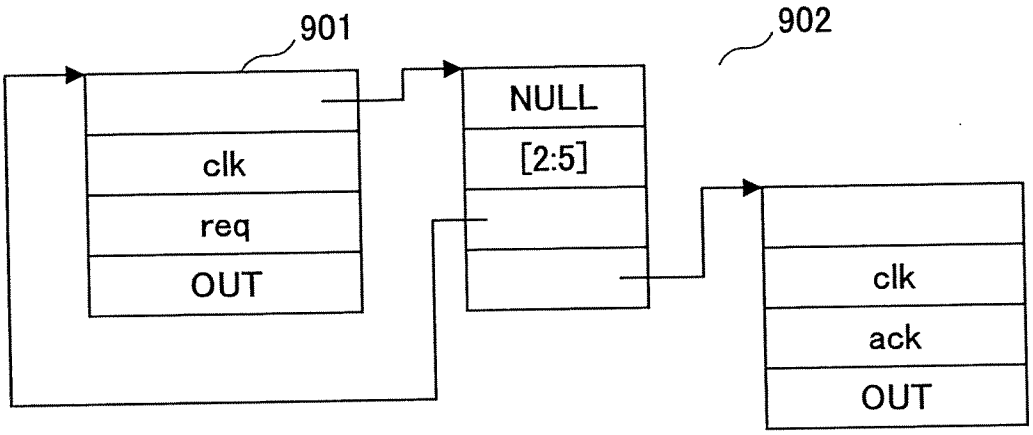
901
902
clk
req
OUT
NULL
[2:5]
clk
ack
OUT

FIG.10

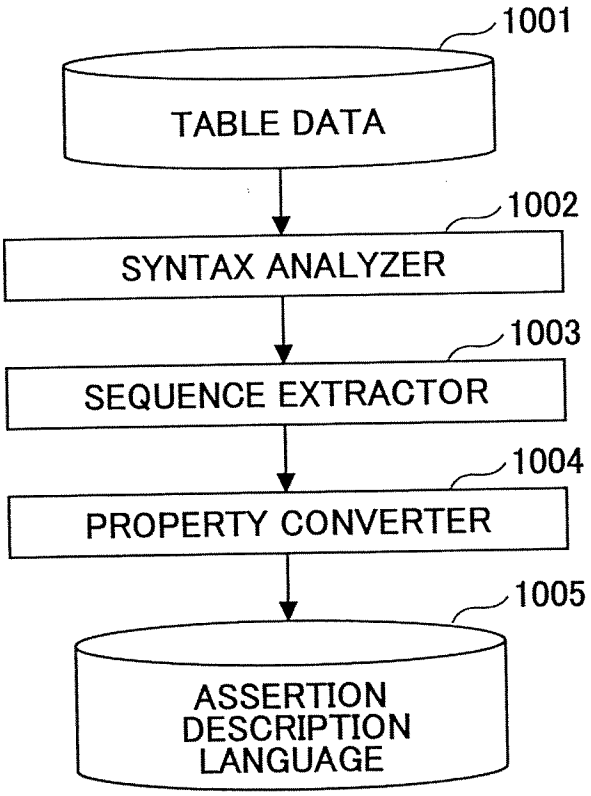
1001
TABLE DATA
1002
SYNTAX ANALYZER
1003
SEQUENCE EXTRACTOR
1004
PROPERTY CONVERTER
1005
ASSERTION DESCRIPTION LANGUAGE

FIG.11

NAME	FADD			
CLOCK	posedge clk			
CONDITION	CONDITION	CONDITION	EXPECT	EXPECT
A	B	CI	S	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

FIG.12

```
typedef struct piece {
    struct piece        *left, *right, *top, *bottom
    char*               str;

} Piece ;
```

FIG.13

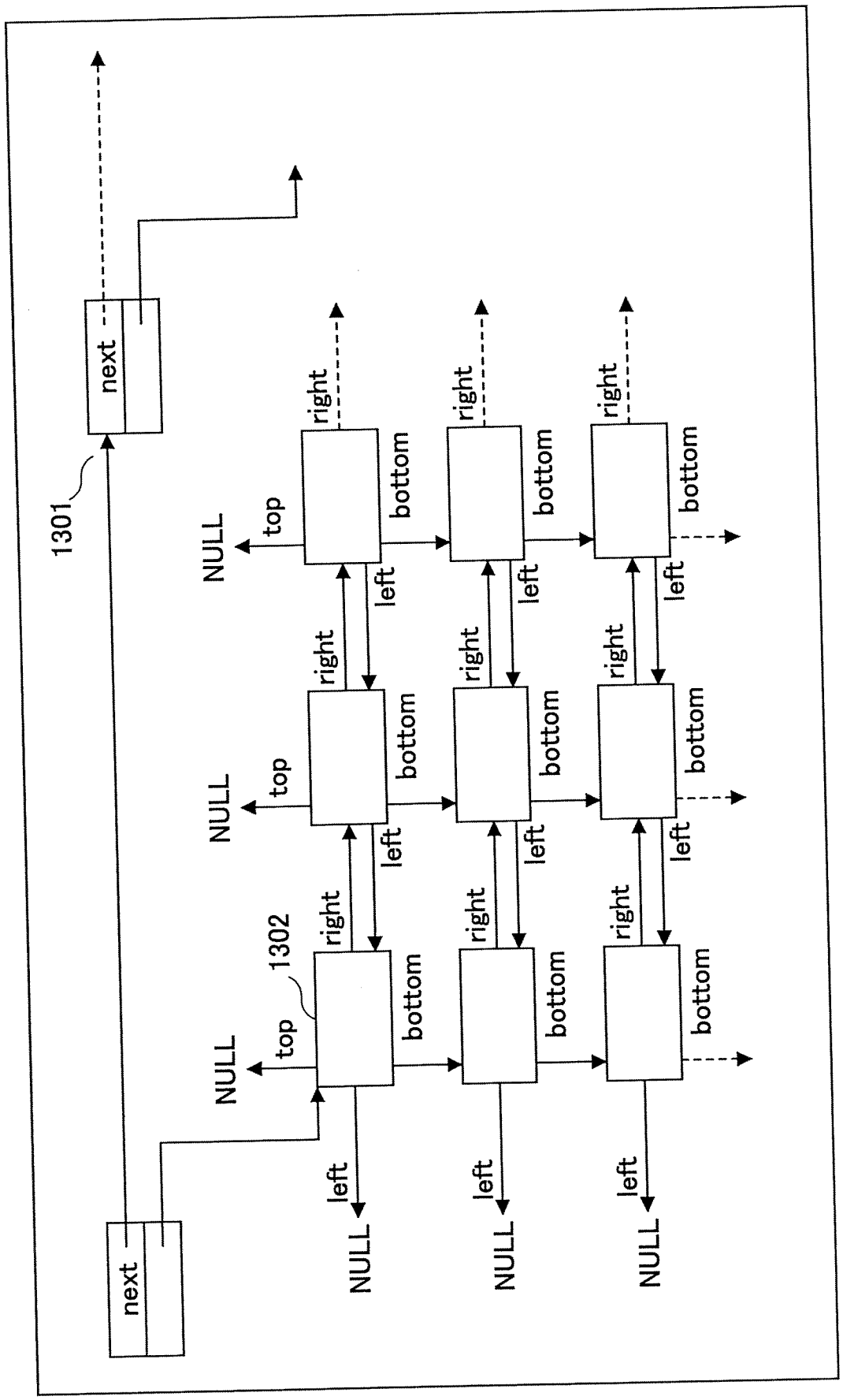

FIG.14

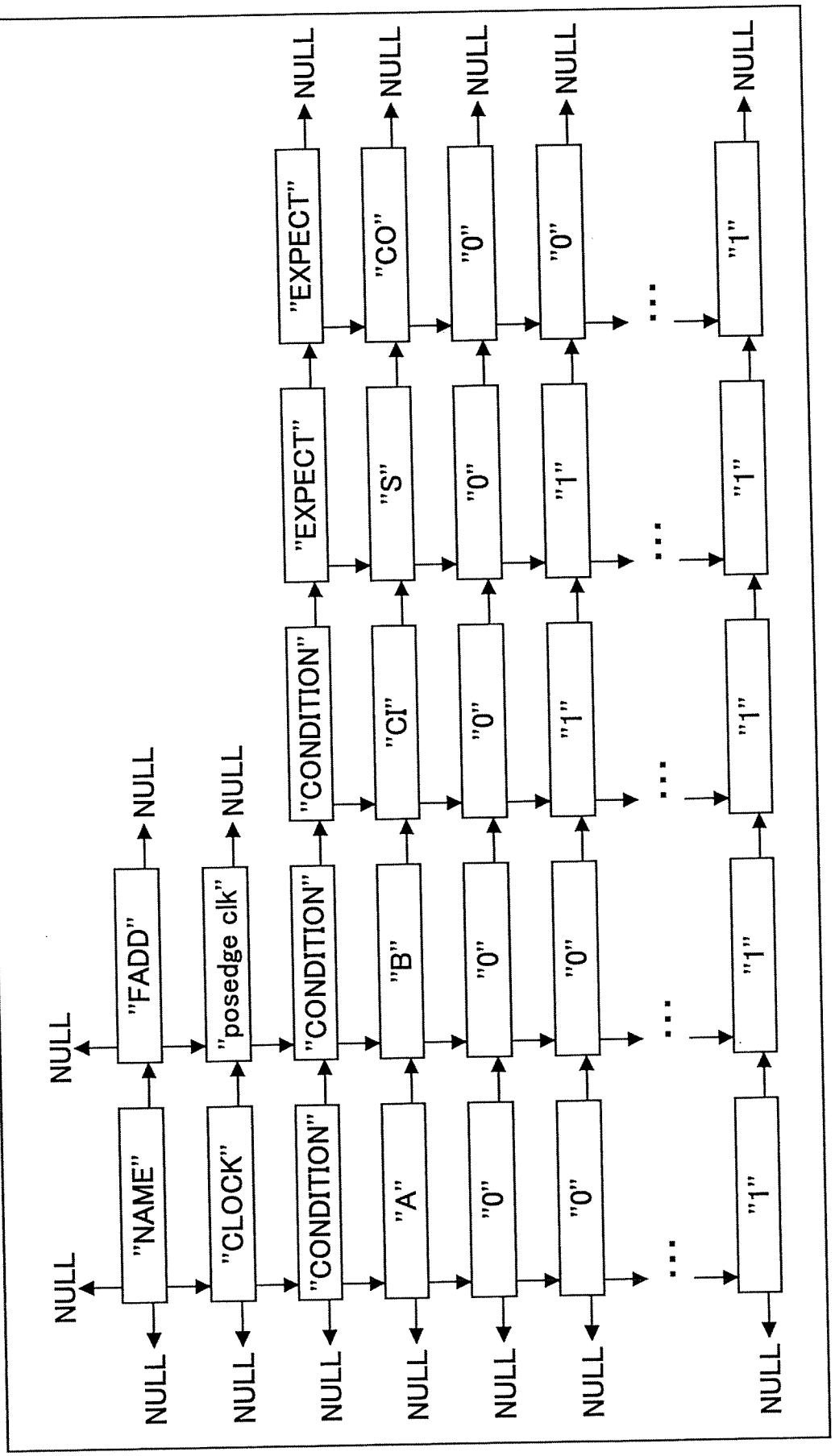
NULL
"NAME"
"CLOCK"
"CONDITION"
"A"
"0"
"0"
"1"
NULL
"FADD"
"posedge clk"
"CONDITION"
"B"
"0"
"0"
"1"
NULL
"CONDITION"
"CI"
"0"
"1"
"1"
NULL
"EXPECT"
"S"
"0"
"1"
"1"
NULL
"EXPECT"
"CO"
"0"
"0"
"1"
NULL

FIG.15

NAME	reqack
CLOCK	posedge clk
CONDITION	EXPECT
req	ack
rose	0[*2:5];1;0